JSP - CUSTOM TAGS

A custom tag is a user-defined JSP language element. When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler. The Web container then invokes those operations when the JSP page's servlet is executed.

JSP tag extensions let you create new tags that you can insert directly into a JavaServer Page just as you would the built-in tags you learned about in earlier chapter. The JSP 2.0 specification introduced Simple Tag Handlers for writing these custom tags.

To write a customer tab you can simply extend SimpleTagSupport class and override the **doTag()** method, where you can place your code to generate content for the tag.

Create "Hello" Tag:

Consider you want to define a custom tag named <ex:Hello> and you want to use it in the following fashion without a body:

```
<ex:Hello />
```

To create a custom JSP tag, you must first create a Java class that acts as a tag handler. So let us create HelloTag class as follows:

```
package com.tutorialspoint;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class HelloTag extends SimpleTagSupport {

  public void doTag() throws JspException, IOException {
    JspWriter out = getJspContext().getOut();
    out.println("Hello Custom Tag!");
  }
}
```

Above code has simple coding where doTag() method takes the current JspContext object using getJspContext() method and uses it to send "Hello Custom Tag!" to the current JspWriter object.

Let us compile above class and copy it in a directory available in environment variable CLASSPATH. Finally create following tag library file: <Tomcat-Installation-Directory>webapps\ROOT\WEB-INF\custom.tld.

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD</short-name>
  <tag>
    <name>Hello</name>
    <tag-class>com.tutorialspoint.HelloTag</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

Now it's time to use above defined custom tag **Hello** in our JSP program as follows:

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
```

```
<html>
  <head>
    <title>A sample custom tag</title>
  </head>
  <body>
    <ex:Hello/>
  </body>
</html>
```

Try to call above JSP and this should produce following result:

```
Hello Custom Tag!
```

Accessing the Tag Body:

You can include a message in the body of the tag as you have seen with standard tags. Consider you want to define a custom tag named <ex:Hello> and you want to use it in the following fashion with a body:

```
<ex:Hello>
   This is message body
</ex:Hello>
```

Let us make following changes in above our tag code to process the body of the tag:

```
package com.tutorialspoint;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class HelloTag extends SimpleTagSupport {

   StringWriter sw = new StringWriter();
   public void doTag()
      throws JspException, IOException
    {
       getJspBody().invoke(sw);
       getJspContext().getOut().println(sw.toString());
    }

}
```

In this case, the output resulting from the invocation is first captured into a StringWriter before being written to the JspWriter associated with the tag. Now accordingly we need to change TLD file as follows:

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD with Body</short-name>
  <tag>
    <name>Hello</name>
    <tag-class>com.tutorialspoint.HelloTag</tag-class>
    <body-content>scriptless</body-content>
  </tag>
</taglib>
```

Now let us call above tag with proper body as follows:

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
  <head>
    <title>A sample custom tag</title>
  </head>
  <body>
    <ex:Hello>
```

```
        This is message body
    </ex:Hello>
  </body>
</html>
```

This will produce following result:

```
This is message body
```

Custom Tag Attributes:

You can use various attributes along with your custom tags. To accept an attribute value, a custom tag class needs to implement setter methods, identical to JavaBean setter methods as shown below:

```
package com.tutorialspoint;

import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class HelloTag extends SimpleTagSupport {

   private String message;

   public void setMessage(String msg) {
      this.message = msg;
   }

   StringWriter sw = new StringWriter();

   public void doTag()
      throws JspException, IOException
    {
       if (message != null) {
          /* Use message from attribute */
          JspWriter out = getJspContext().getOut();
          out.println( message );
       }
       else {
          /* use message from the body */
          getJspBody().invoke(sw);
          getJspContext().getOut().println(sw.toString());
       }
   }

}
```

The attribute's name is "message", so the setter method is setMessage(). Now let us add this attribute in TLD file using <attribute> element as follows:

```
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>Example TLD with Body</short-name>
  <tag>
    <name>Hello</name>
    <tag-class>com.tutorialspoint.HelloTag</tag-class>
    <body-content>scriptless</body-content>
    <attribute>
       <name>message</name>
    </attribute>
  </tag>
</taglib>
```

Now let us try following JSP with message attribute as follows:

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
  <head>
    <title>A sample custom tag</title>
  </head>
  <body>
    <ex:Hello message="This is custom tag" />
  </body>
</html>
```

This will produce following result:

```
This is custom tag
```

Hope above example makes sense for you. It would be worth to note that you can include following properties for an attribute:

Property	**Purpose**
name	The name element defines the name of an attribute. Each attribute name must be unique for a particular tag.
required	This specifies if this attribute is required or optional. It would be false for optional.
rtexprvalue	Declares if a runtime expression value for a tag attribute is valid
type	Defines the Java class-type of this attribute. By default it is assumed as **String**
description	Informational description can be provided.
fragment	Declares if this attribute value should be treated as a **JspFragment**.

Following is the example to specify properties related to an attribute:

```
.....
    <attribute>
      <name>attribute_name</name>
      <required>false</required>
      <type>java.util.Date</type>
      <fragment>false</fragment>
    </attribute>
.....
```

If you are using two attributes then you can modify your TLD as follows:

```
.....
    <attribute>
      <name>attribute_name1</name>
      <required>false</required>
      <type>java.util.Boolean</type>
      <fragment>false</fragment>
    </attribute>
    <attribute>
      <name>attribute_name2</name>
      <required>true</required>
      <type>java.util.Date</type>
    </attribute>
.....
```